

Report for Udacity Machine Learning Nanodegree (MLND); Capstone Project: Create a Customer Segmentation Report for Arvato Financial Services



Clear Creek in Winslow, AZ

Prepared by: Amiri McCain

October 28, 2019

Project Overview

The objective of this project is to analyze demographics data of customers of a mail-order sales company in Germany that sells organic products and compare that to demographics data of the general population of Germany. The end goal is to be able to predict, based on demographics data, which individuals from the general population should be targeted in the mail-order campaign.

Both unsupervised and supervised learning techniques will be used. Unsupervised learning will be used to help identify segments of the general population of Germany that best matches the existing customer base of the company. A supervised learning prediction model will be developed to predict the likelihood of whether or not an individual of the general population will become a customer. The dataset was provided by a real business - Bertelsmann Arvato Analytics and represents a real-life data science task.

This project was provided by Udacity in cooperation with Arvato Financial Services. The final Jupyter Notebook can be found on GitHub at: <https://github.com/AmiriMc>. However, the datasets provided for this project, courtesy of Arvato Financial Services, are private and cannot be made public.

Completion of this capstone project includes four parts:

1. Create a customer segmentation report for Arvato Financial Solutions.
2. Create a supervised learning model and verify this prediction model with training data.
3. Submit the prediction model to the “Udacity+Arvato: Identify Customer Segments” Kaggle competition for testing [here](#).
4. Create a report detailing my analysis and results.

Domain Background

Arvato Bertelsmann Financial Solutions is an IT service management company headquartered in Baden-Baden, Germany that specializes in optimizing the financial backbone of businesses and provides professional B2B (business-to-business) financial services. Arvato provides expertise in automation and data analytics that provide a comprehensive overview of an entire business and the business’s client’s journey.

Problem Statement

Arvato Financial Solutions wants their client to be able to analyze attributes to identify payment behavior, predict payment behavior, recommend credit scores and calculate credit scores of their clients. This will allow Arvato’s client to more efficiently target the right type of clients for their organic mail order catalog. This appears to be a clustering and classification problem.

The list below indicates how the results of the customer segmentation analysis report will be used and the questions Arvato’s client desires to answer:

- How can the client, a mail order company acquire new clients more efficiently for its organic mail order catalog?
- The mission of the engagement with the client is to make decisions based on data instead of gut feel.
- Arvato’s client will be using resultant datapoints to improve decisions and show rational behind decisions.
- Arvato wants their client to use and request reports more often.

Datasets

There are four data files connected with this project:

- `Udacity_AZDIAS_052018.csv`: Demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns).
- `Udacity_CUSTOMERS_052018.csv`: Demographics data for CUSTOMERS of a mail-order company; 191 652 persons (rows) x 369 features (columns).
- `Udacity_MAILOUT_052018_TRAIN.csv`: Demographics data for individuals who were targets of a marketing campaign; 42 982 persons (rows) x 367 (columns).
- `Udacity_MAILOUT_052018_TEST.csv`: Demographics data for individuals who were targets of a marketing campaign; 42 833 persons (rows) x 366 (columns).

In addition to the four data files listed above, two Excel spreadsheet files were included to provide more information about the columns (features) depicted in the data files. The file names are “DIAS Attributes - Values 2017.xlsx” and “DIAS Information Levels - Attributes 2017.xlsx.” These Excel files provide additional information, such as descriptions of attributes and a detailed mapping of data values for each feature and in alphabetical order, about the columns in the primary data files.

Evaluation Metrics

The prediction model predicted which people are most likely to respond to the advertising that they will receive through the mail. In addition to evaluating this model with training data, this prediction model was also submitted to the “Udacity+Arvato: Identify Customer Segments” Kaggle competition. The evaluation metric that Kaggle uses for this competition is the Area Under the Curve (AUC) for the Receiver Operating Characteristic (ROC) curve. The AUC ROC is immune to class imbalance and this dataset will be highly imbalanced.

The LGBMRegressor classifier was used for the final score. My top Kaggle score was 0.79542. The Leaderboard can be viewed at the link: <https://www.kaggle.com/c/udacity-arvato-identify-CUSTOMERS/leaderboard>.








#	Team Name	Notebook	Team Members	Score ?	Entries	Last
1	TensorFrozen(Shihao)			0.80819	29	1y
2	Gaurav Ansal			0.80816	27	3mo
3	Michel N			0.80777	24	9mo
...						
30	Tobias Gorgs			0.79627	3	5mo
31	Amiri			0.79542	31	~10s
Your Best Entry ↑ You advanced 14 places on the leaderboard! Your submission scored 0.79542, which is an improvement of your previous score of 0.69638. Great job! Tweet this!						
32	bvcm			0.79475	4	3mo
33	jaouad			0.79411	1	6mo

Figure 1: Kaggle competition public Leaderboard score.

Data Exploration, Preprocessing and Cleaning

Taking some time to explore the data, it became readily apparent that there are numerous empty data cells throughout; hundreds of thousands, in both the AZDIAS and CUSTOMERS datasets. The AZDIAS dataset had 33,492,923 empty cells and the CUSTOMERS dataset had 13,864,522 empty cells. That is a lot of missing data! All of this missing data certainly will have an impact on the analysis and so the data needs to be massaged to accommodate these huge gaps.

There were also a few columns that were coded with letters, words, and numbers (alphanumeric), those were columns: `OST_WEST_KZ`, `CAMEO_DEUG_2015` and `CAMEO_INTL_2015`. The “columns” are synonymous with features – *each column is a feature of the dataset*. One hot encoding was used to encode features with categorical variables into numerical variables. In the case of `OST_WEST_KZ` I manually created the one hot encoding functionality, in the case of `CAMEO_DEUG_2015` and `CAMEO_INTL_2015`, Scikit-learn’s `LabelEncoder` and `OneHotEncoder` functions were used. These particular columns at times had ‘X’ and ‘XX’ in them. This was not discovered this until the predictor was ran later in the project. I then had to go back and update the data cleaning function to handle this. There was also a feature named `EINGEFUEGT_AM` that was coded with a date/time stamp, this feature was deleted.

```

1 print(azdias.shape)
2 azdias.head()

```

(891221, 366)

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTI
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0	
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	

5 rows x 366 columns

Figure 2: A look at the AZDIAS and CUSTOMERS dataset readily shows many empty data cells.

Initially, `df.dropna(axis=1)` was used to quickly drop all columns with any missing data. That resulted in 93 columns remaining out of 366 (AZDIAS). The clustering analysis of the remaining 93 features for both AZDIAS and CUSTOMERS showed that the two clusters looked quite similar to each other and the supervised learning model did not perform well, even after extensive hyperparameter tuning. These two factors indicate that too many features had been deleted and there wasn't enough variance in the remaining features neither to create sufficiently unique clusters, nor make accurate predictions.

At this point, it was necessary to go back and do more analysis and much better precleaning, processing, feature selection and feature engineering. Functions were created that would search columns and rows for a predetermined percentage of missing values and then drop those columns and rows if they had a higher percentage of missing data than the predetermined input value.

It was helpful to download the datasets locally for viewing in Excel – it is beneficial to be able to peruse the data in this way to help one get a better overall sense of the dataset.

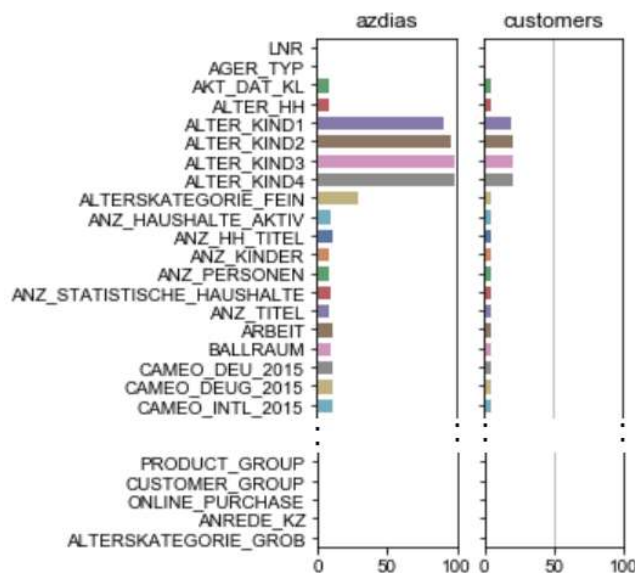


Figure 3: Graphs listing percentage (x-axis) of missing data for each feature.

Highlights of the data preprocessing and cleaning step:

- Replaced all empty cells with `NaN`.
- A 30% threshold seemed to be optimal for columns and rows; thus, any column or row with more than 30% missing data was dropped.
- Dropped 6 columns.
- Dropped 140,637 rows for AZDIAS; 56,608 rows for CUSTOMERS.
- Replaced remaining empty cells with most occurring value for that column (feature).
- Applied one hot encoding for: `OST_WEST_KZ`, `CAMEO_DEUG_2015` and `CAMEO_INTL_2015`.
- Made sure AZDIAS and CUSTOMERS had the same shape and features after processing.
- Final number of features: 358

```

1 %%time
2 azdias_clean = data_preprocess_clean(azdias, operation='both', delete_rows=missing_data_rows_az)

- processing columns
rows, columns (before): (891221, 366)
rows, columns (after): (891221, 360)
- processing rows
rows, columns (before): (891221, 360)
rows, columns (after): (750584, 360)
- filling empty cells with most frequent value for each column
- one hot encoding for OST_WEST_KZ and CAMEO_DEU_2015
- replacing "X" and "XX" in CAMEO_DEU_2015 column w/most frequent of column
- replacing "X" and "XX" in CAMEO_DEUG_2015 column w/most frequent of column
- replacing "X" and "XX" in CAMEO_INTL_2015 column w/most frequent of column
Wall time: 3min 36s

```

Figure 4: Example of calling the `data_preprocess_clean` function and the functions dialog; returns a clean dataframe (df).

It is important to explore the data and then to preprocess and clean the data. This was a very time consuming and involved process, not to mention quite challenging at times. Some of the functions and models that will need to be used cannot handle missing data or data with letters and words, functions such as k-means and most, if not all, classifiers will throw errors when you try and pass empty data or alphabetical data into it, one reason is that these functions convert the data to floating point numbers before running and you cannot convert an empty cell or letters to a floating point number!

I experimented with dropping rows and columns based on varying amounts of missing data, for example; 5%, 10%, 20%, and 40%. At 5% and 10% threshold, apparently too much data was removed and the predictors performed poorly. At 40% and above, the predictor performance started to drop ever so slightly or showed no improvement and going from 20% to 30%, the predictor performance improved – 30% seems to be the optimal number in this case.

Data Visualization

After exploration, preprocessing and cleaning, it is important to perform some data visualization. Since there were still 358 remaining features, all of them, for the sake of time, cannot be analyzed; however, a sampling of them can. Ten different features were selected to create histogram plots and compare AZDIAS and CUSTOMERS. Most feature histograms showed that the data between AZDIAS and CUSTOMERS were quite different. For example, the most important feature `D19_SOZIALES`, which happened to not be described anywhere in the “DIAS” spreadsheets, showed that “0” (zero) was the most occurring value for the general population of Germany, but 1 was the most occurring value for CUSTOMERS.

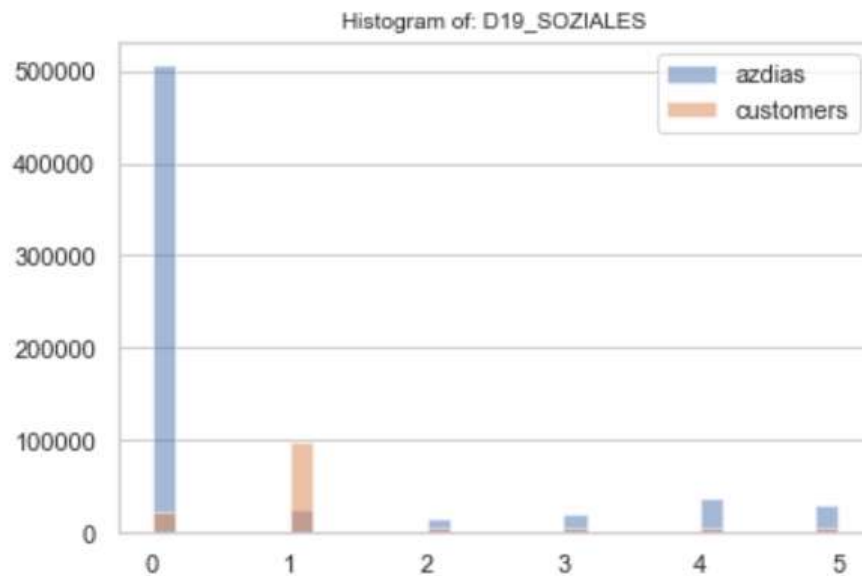


Figure 5: Histogram plot of feature 'D19_SOZIALES'. This was the most important feature in the analysis.

Next and before Principal Component Analysis (PCA) or clustering, the data was scaled using: ``StandardScaler().fit_transform(df)``. This is imperative for k-means clustering functions so that all the features are on the same scale; otherwise, features with large value ranges can skew the appearance of the graphs and plots, making them difficult to read and comprehend. PCA is also sensitive to the scaling of the feature variables.

CUSTOMER SEGMENTATION REPORT

Principal Component Analysis (PCA) and K-Means Clustering

PCA converts a set of possibly correlated data into a set of values of linearly uncorrelated variables called principal components. It is a technique for dimensionality (or feature) reduction. The components that are created are such that the first component from the results, has the most variation or impact on the dataset. Each component that is created can be plotted against the weighting of the features that make up that component – the component that comes about as a result of the analysis can be thought of as a new feature, at times you may even wish to give the component a new feature name, after carefully analyzing its makeup.

PCA was used for dimensionality reduction (reduce the number of features). For PCA, all 358 features were used, this was so that it could easily be observed from the [explained variance ratio](#), how many features it takes to make up 80% or 90%+ of the variance.

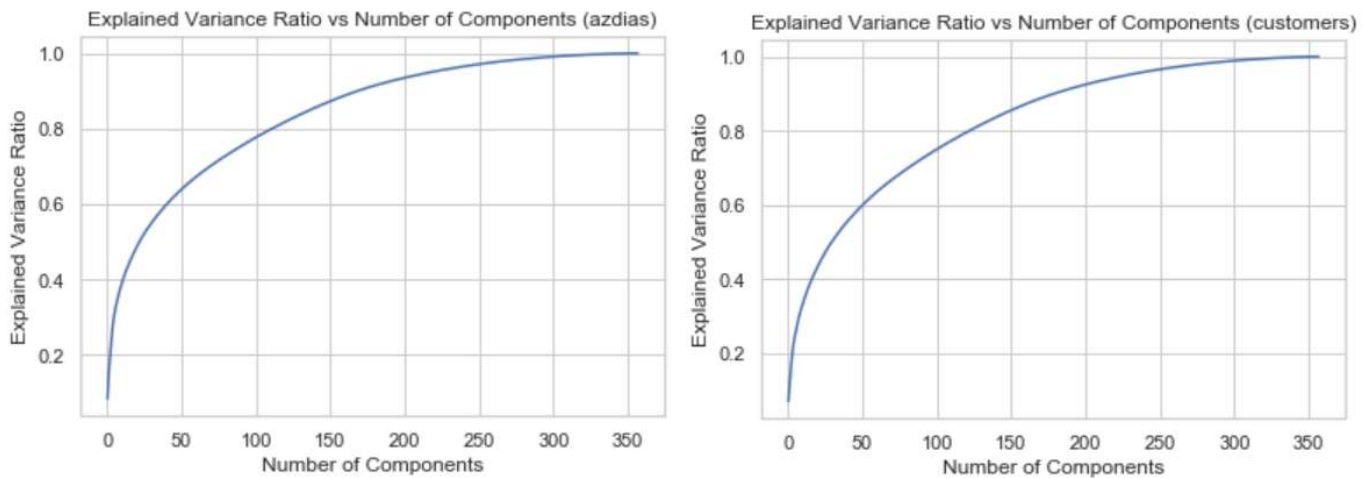


Figure 6: Explained Variance Ratio comparison between AZDIAS and CUSTOMERS.

From the comparison of two graphs in figure 6, it can be understood that at 200 components, more than 90% of the variance of the data is accounted for and so in theory, we should be able to use just 200 components, instead of 358, to create a robust predictor.

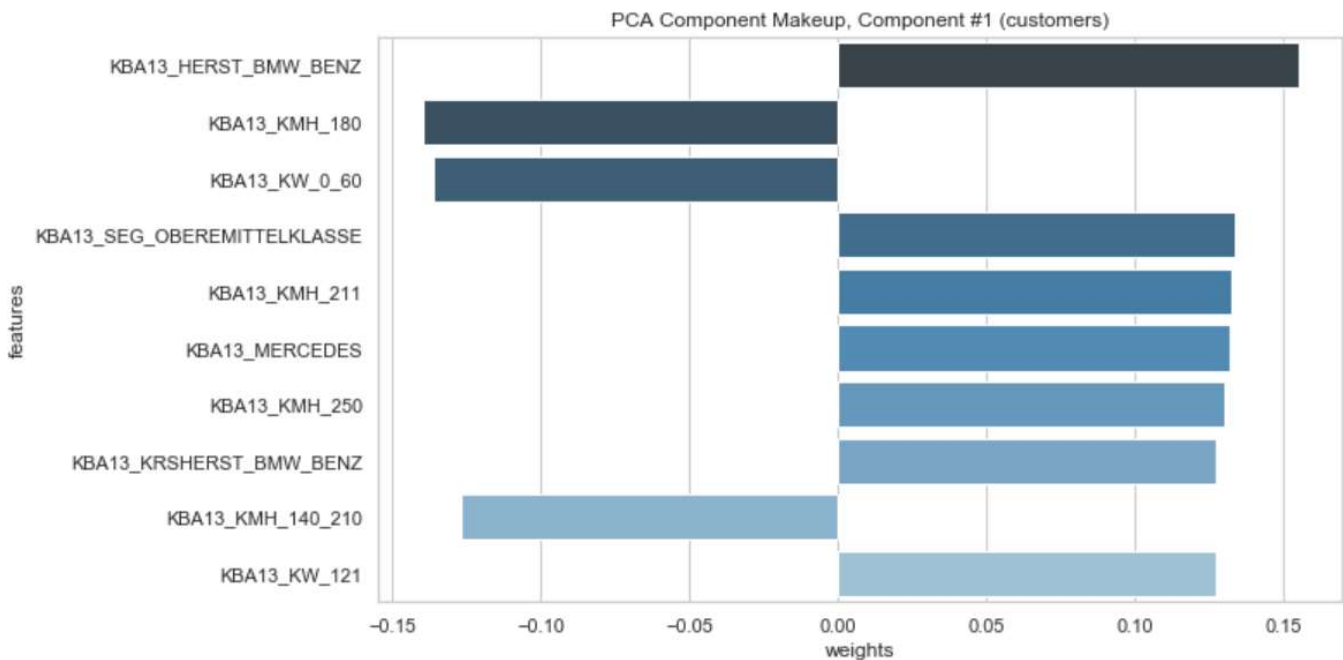


Figure 7: PCA component makeup graph.

Figure 7 shows the feature weighting for each feature for component 'n'. In the figure, component number 1, the component with the most variance was plotted. As mentioned earlier, the generic name of "Component #1" could be renamed to something more descriptive, if the makeup of this component is sufficiently understood. For example, in the figure, it looks like features having to do with automobiles are positively correlated (results are >0) with this component; whereas, all of the features that are weighted to the left of zero (results <0) are negatively correlated (less variance) with the component. With this new information, we could rename

component one to “high end auto and horsepower” (e.g. “HE_AUTO_HP”). Doing this analysis on the top components helps to identify important features that we want to be sure to include in our final model.

Unsupervised Learning Using K-Means Clustering

K-means clustering is an unsupervised learning technique. This will typically be used when labeled (classified) data is not available and you are trying to group the data into like clusters. Multiple K-Means functions were created during this process. The first K-Means function was created to [fit](#) a K-Means model based on a single number of clusters. The second K-Means function that was created plotted the K-Means scoring data across numerous numbers of clusters. This was so that the “knee” (bend) in the plot could be found. The knee is generally an indicator of the value you pick for your number of clusters. For purposes of comparison with the two charts below, ignore the y-axis scaling.

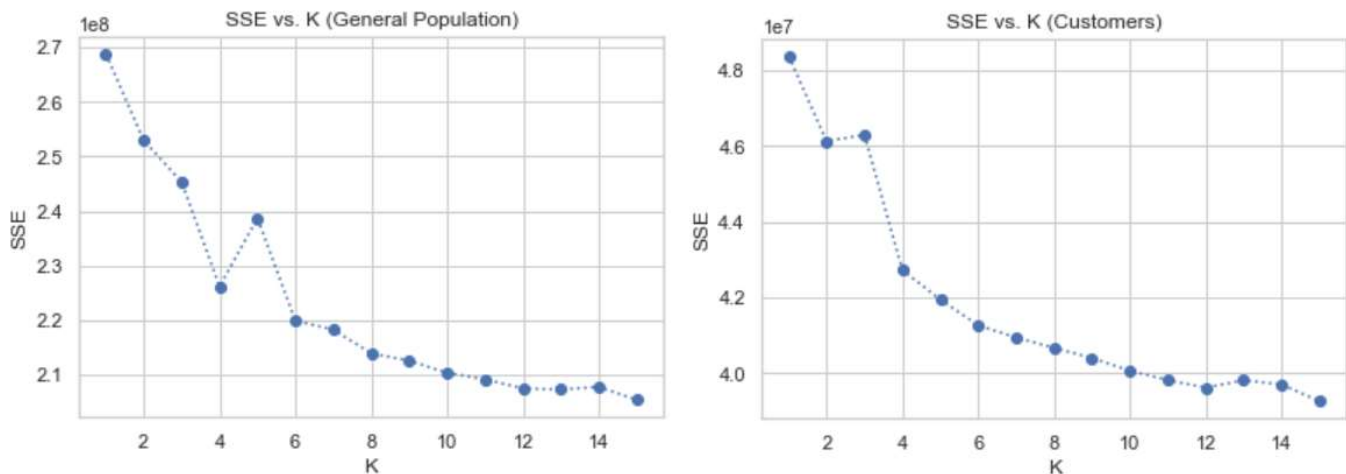


Figure 8: K-Means plot across “n” number of clusters in order to find the knee.

For K-Means, Scikit-learn’s [MiniBatchKMeans](#) was used for clustering. MiniBatchKMeans performs particularly well on large datasets and ran probably 100 times faster than the classic implementation of K-Means. Considering the numerous iterations that were ran, speed was needed! MiniBatchKMeans does incremental updates of the centers using mini-batches, instead of consuming the whole dataset at once the way standard [K-Means](#) does and not much accuracy is lost. In Figure 8, it is somewhat difficult to see the textbook knee in the data, the data seems a bit noisy, perhaps this was due to the use of MiniBatchKMeans.

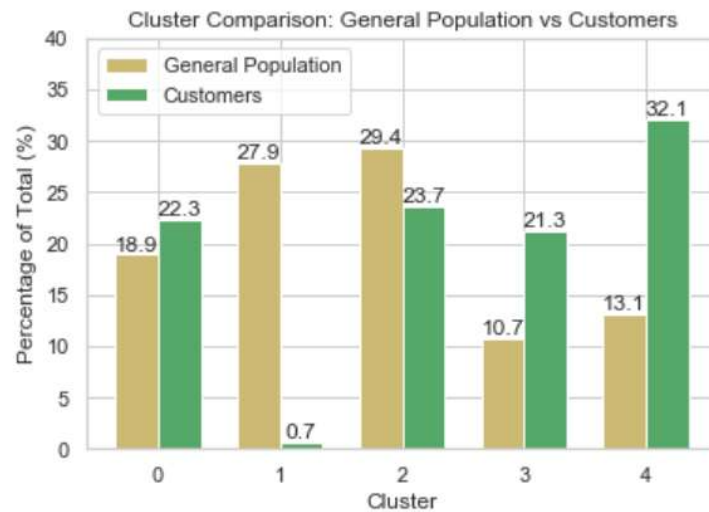


Figure 9: Cluster comparison of general population (AZDIAS) to CUSTOMERS.

Five clusters were used for the analysis. Then, the five clusters were graphed for both the general population data (AZDIAS) and the CUSTOMERS data as a percentage of total. The function that generated the data in Figure 9 created an array of cluster count label occurrences that correspond to the given dataset, so in other words, 18.9% of the general population data was put into group or cluster 0; whereas, only 0.7 percent of the CUSTOMERS population was put into group or cluster 1. Ideally, the general population and CUSTOMERS values should be different for each cluster – this helps us to gain valuable insight. What do we learn from this graph? For clusters 1 and 2, the general population is overrepresented, we probably do not want to target any of the general population in these clusters, especially in cluster 1 since only 0.7 of actual CUSTOMERS belong to this group – our goal is to target clusters of people where the customer representation is high in that cluster and that preferably the general population in that same cluster is underrepresented.

SUPERVISED LEARNING

Turning all of the useful preprocessing and cleaning code into functions in the beginning of the project, made this section easier, as the functions for preprocessing and cleaning could be called in a simple one or two lines of code. In this section, a number of classifiers were experimented with in order to target the appropriate demographics so that the highest ROC AUC score could be obtained. The higher the score, the more of the general population were correctly identified as potential customers, as they responded to the mailout flyer.

Table 1: List of technologies used and descriptions

Technology	Description	Link
RandomForestClassifier()	Meta estimator that fits a number of decision tree classifiers.	Link
XGBClassifier()	An efficient gradient boosting classifier.	Link
GradientBoostingClassifier()	GB builds an additive model in a forward stage-wise fashion.	Link
LGBMRegressor()	Gradient boosting framework that uses tree-based learning.	Link
GridSearchCV()	Performs an exhaustive search over specified parameters for a given classifier, returns the best combination of parameters.	Link
SMOTE()	Synthetic Minority Oversampling Technique.	Link

Technology	Description	Link
RandomUnderSampler()	Undersamples the majority class(es); e.g. randomly removes majority class data points (rows).	Link
NearMiss()	Class to perform under-sampling based on NearMiss methods.	Link
SMOTETomek()	Combine over and under sampling using SMOTE and Tomek links.	Link
StratifiedKFold()	For cross validation, provides train/test indices and can be used in conjunction with fit or predict and return 'n' numbers of predictions. This method works well for highly imbalanced datasets. Folds are made by preserving the percentage of samples for each class.	Link
train_test_split()	Split dataset into random train and test subsets.	Link
Confusion matrix	For statistical classification, allows visualization of the performance of a supervised learning algorithm.	Link
ROC AUC	Receiver Operating Characteristic – Area Under the Curve; graphical plot of the ability of a binary classifier. The plot consists of the true positive (TP) rate (y-axis) plotted against the false positive (FP) rate (x-axis). Area under this curve is then calculated.	Link

The `mailout_train` data was ran through the `data_preprocess_clean()` function. The random forest classifier was tried out first. My initial Kaggle score was a measly 0.48854 (or 48.85%), which is basically worse than randomly guessing the classification (good potential customer (1) or not (0)). My score improved by about 6% when CUSTOMERS data was added – I added data from the CUSTOMERS dataset so that customer and non-customer data were balanced in the number of samples each had. This seemed like an ingenious idea and I immediately saw improvement. At this point, I tried different classifiers such as XGBoost(), GradientBoostingClassifier(), and LGBMRegressor(). The LGBMRegressor actually performed the worst in this scenario, even though this is the classifier I ended up using for my final score. I also tried different undersampling and oversampling techniques, as listed in the table above, to no avail. At times, there were very minor improvements, but nothing that showed real promise.

The reason it was difficult to classify this problem is because the dataset is highly [imbalanced](#). When the `mailout_train` data is examined, it can be seen that out of a total of 34,184 points, only 424 of those points are customers. Looked at another way, only 1.24% of the dataset are customers, that is an 80:1 ratio. If the preprocessing and cleaning are not done properly and or the classifier is not tuned properly, it is easy for the classifier to classify everything as “0” (non-customer).

The real breakthrough came when I went back and made the preprocessing and data cleaning functionality more robust and then implemented `GridSearchCV` to programmatically test numerous combinations of patterns – also known as hyperparameter tuning. I almost passed up using grid search since I felt like I had already run so many different iterations already, but in the end, no data was used from the CUSTOMERS dataset and no under/over sampling was used – these made the results worse, at least for LGBMRegressor. Learning rate, number of estimators and max depth were the parameters I chose to manipulate (see Figure 10). I noticed that every time that the size of the dataset changed – either columns or rows, grid search returned different optimal parameters. I would not have guessed that slightly changing the number of rows would have much of an effect on the optimal parameters, but it did. I made sure that the parameter selections that grid search chose were never in the upper or lower corner, I always adjusted it to make sure it had another option. For example, looking at `param_grid` in Figure 10, when 70 estimators was the upper limit for the search and the grid search returned 70 as the selection, I added 80, 90 and then reran the search to be sure that grid search had the opportunity to check out 80 or 90 estimators; otherwise, if I left 70 as the highest, I would not know for sure if that 70 was the optimal if I never checked it out.

```

1 param_grid = {
2     'learning_rate': [0.04,0.05,0.06],
3     'n_estimators': [60, 70, 80],
4     'max_depth': [1, 2, 3]
5 }
6
7 from lightgbm import LGBMRegressor
8 from sklearn.model_selection import GridSearchCV
9
10 clf = LGBMRegressor(n_estimators=10)
11 clf = GridSearchCV(clf, param_grid, cv=3)
12
13 start = time.time()
14 clf_fit = clf.fit(train_features, train_labels)
15 end = time.time()
16 print('wall time (s): ', end - start)
17 print(clf_fit)
18 print()
19
20 print('Best parameters found by grid search are:', clf.best_params_)
21 print('Best score found by grid search are:', clf.best_score_)
22

```

Figure 10: GridSearchCV() code. Greatly helped to improve classifier predictions and Kaggle score.

The optimal classifier turned out to be LGBMRegressor and the optimal parameters turned out to be:

- LGBMRegressor(learning_rate=0.05, max_depth=2, n_estimators=70)

Figure 11 shows ten features (out of 358) listed from most important to least important:

	Feature	Importance
64	D19_SOZIALES	56
54	D19_KONSUMTYP_MAX	26
162	KBA05_SEG2	16
347	VHA	6
13	CAMEO_DEU_2015	6
340	STRUKTURTYP	6
5	ANZ_HAUSHALTE_AKTIV	6
3	ALTER_HH	5
332	SEMIO_PFLICHT	5
126	KBA05_CCM4	4

Figure 11: Top 10 most important features, based on LGBMRegressor().

Model metrics and confusion matrix:

- True Positives (TP): 0
- False Positives (FP): 0
- False Negatives (FN): 117
- True Negatives (TN): 10,139

Table 2: Confusion matrix

n = 10,256	Predicted: 0	Predicted: 1
Actual: 0	10,139	0
Actual: 1	117	0

Most of my confusion matrices did not look like this, but the one classifier configuration that gave me the best results had zeros in the right-hand column, go figure. It is interesting that there were no true positives, but also no false positives. There were; however, 117 false negatives, which says that is wrongly classified all of the true positives as zeros. So, from the confusion matrix, our model did not predict any positives (1's), only negatives (0's).

The AUC score for the LGBMRegressor seemed to most closely match the Kaggle score which was encouraging. The AUC score for the other classifiers that were used did not match as closely to the Kaggle score.

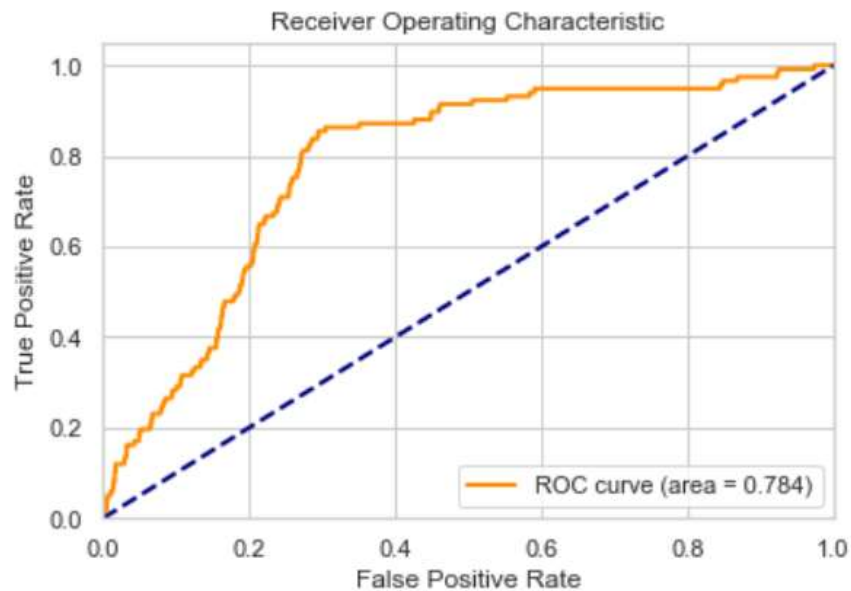


Figure 12: ROC AUC operating characteristic curve and AUC score. The blue dashed line represents an AUC of 50%.

Ideally, the first rising slope of the curve should be as upwardly steep as possible. This means that the classifier is capturing a large amount of the true customers. ROC AUC is also immune to class imbalance, for the most part and it is the method Kaggle uses to score all entries.

- ✓ Final Kaggle score: **0.79542** (difference of +1.142 percent between score in Figure 12 and Kaggle).

The Kaggle entry was a two-column csv file, the first column “LNR” is an ID column and the second column, “RESPONSE” is for the predictions. However, RESPONSE is a statistical probability, not a straight 0 or 1 prediction.

Areas for future improvement

This was an exciting and engaging project to work on, lots of up and downs and I was pleased to get nearly an 80% score for the Kaggle competition. Nonetheless, I documented several areas for improvement that has the high potential to produce a higher Kaggle score. After all, the bottom-line important goal for Arvato is to help their client target the correct individuals of the general population of Germany so that these too, can become customers of the client’s organic mail order catalog and so the higher the score, the better.

Feature engineering/selection for experimentation and improvement:

- Remove features that have high amounts of unknowns, represented in data as “-1” or “0”.
- Reengineer original alphanumeric coded features such as CAMEO_DEU_2015 and PRAEGENDE_JUGENDJAHRE into multiple features for more granularity.
- Remove features with too many categories, such as CAMEO_DEUINTL_2015 and LP_LEBENSPHASE_FEIN.
- Take a deeper dive into the data and find useful relationships between features that provide insight. This may allow us to create new meaningful features with the data.
- Delete other features that are not useful. Some features do not seem that relevant and so it would be a good experiment to selectively remove these features and then document how they affect the predictor outcomes.
- Better application of PCA and K-Means.

References

1. Arvato Financial Solutions: <https://finance.arvato.com/en-us/>
2. Who Buys Organic Food: Different Types of Consumers: <https://www.thebalancesmb.com/who-buys-organic-food-different-types-of-consumers-2538042>
3. How Germany is leading Europe’s organic food revolution: https://elpais.com/elpais/2017/08/28/inenglish/1503913626_981017.html
4. Americans’ views about and consumption of organic foods: <https://www.pewresearch.org/science/2016/12/01/americans-views-about-and-consumption-of-organic-foods/>
5. Best Practices for Feature Engineering: <https://elitedatascience.com/feature-engineering-best-practices>
6. The Right Way to Oversample in Predictive Modeling: <https://beckernick.github.io/oversampling-modeling/>
7. An Implementation and Explanation of the Random Forest in Python: <https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76>
8. 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset: <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>

9. Train Test Split vs K Fold vs Stratified K fold Cross Validation: <https://www.youtube.com/watch?v=fKz-SgScM3Q>
10. How to One Hot Encode Sequence Data in Python: <https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>
11. What is LightGBM, How to implement it? How to fine tune the parameters?:
<https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc>
12. In-Depth: Decision Trees and Random Forests:
<https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html>
13. In Depth: Principal Component Analysis: <https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html>
14. In-Depth: k-Means Clustering: <https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html>
15. Investigating Customer Segmentation for Arvato Financial Services:
<https://medium.com/@shihaowen/investigating-customer-segmentation-for-arvato-financial-services-52ebcfc8501>
16. 2 Ways To Use Machine Learning to Identify customers: <https://medium.com/@ursulahneumann/2-ways-to-use-machine-learning-to-identify-CUSTOMERS-d33c5c2eaf60>
17. Receiver operating characteristic: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
18. Simple guide to confusion matrix terminology: <https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/>

The list above could probably be hundreds of lines long if I were to list every single thing I referenced over the course of this project. Some items that I referenced I read in-depth and completely - sometimes multiple times, others I “surgically extracted” only the specific details that I needed. There is a wealth of information out there related to every aspect of software development and engineering – the analogy of trying to drink from a firehose comes to mind. Instead of listing them all, I want to express my sincere gratitude to the entire community, especially those who are eager to share and who take the time to do it well – thank you!